http://www.digimedia-mls.com

DMXPLUS.DLL

Win32 DLL for Digimedia DMX-512 interface products

OEM User's Manual
Ver 1.1

©1998 Digimedia Multimedia Lighting Solutions
All rights reserved

TABLE OF CONTENTS

1 - INTRODUCTION 3

2 - DLL FUNCTION DEFINITIONS 3

2.1 - DMXINIT 4

2.2 - DMXDRIVE 5

2.3 - DMXRELAX 6

2.4 - DMXSTANDARD 7

2.5 - DMXSENSE 8

2.6 - DMXCAPTURE 9

2.7- DMXSEND 10

2.8 - DMXPTCSND 11

2.9 - DMXIDENT 13

2.10 - DMXREV 14

2.11 - LIBVER 15

3 - USING DMXPLUS.DLL FUNCTIONS 16

3.1 - Simple Master-only program 16

3.2 - Master-only program with identification and patch-send 17

3.2 - Master-Slave program 18

APPENDIX A – USING DMXPLUS.DLL IN DIFFERENT LANGUAGES 19

A.1 – Delphi 19

A.2 – Visual C++ 23

A.3 – Visual Basic 24

1 - Introduction

DMXPLUS.DLL is a Win32 compatible Dynamic Link Library (DLL) that supports the operation of DMX interfaces offered by Digimedia, namely DMXPlus and DMXAdp.
As with any Win32 DLL, the library may be used by all high-level language environment including the modern RAD environments, operating under Windows95.
Use of DMXPLUS.DLL for use out of Digimedia proprietary lighting control software is subject to an OEM agreement between Digimedia and the OEM customer.

2 - DLL function definitions

Following the Win32 definition, DMXPLUS.DLL uses the **stdcall** calling convention for all parameters.
The list of functions supported by DMXPLUS.DLL is reported in the following table

DMXINIT (lpt,dmxsize,dmxmst)
DMXDRIVE(lpt,dmxstart,dmxstatus)
DMXRELAX(lpt)
DMXSTANDARD(lpt)
DMXSENSE(lpt,dmxstatus,dmxtype)
DMXCAPTURE(lpt,dmxstart,dmxpkt,dmxchan)
DMXSEND(lpt,dmxchan)
DMXPTCSND(lpt,mast,dmxchan,dmxchan1,dmxchan2,dmxchan3,dmxchan4, dmxgr,dmxpatch,dmxpatchlev,dmxcurve)
DMXIDENT(lpt,dmxstatus,dmxhard,dmxrev,dmxsubrev,serialno)
DMXREV(lpt,revsiz,revbuf)
LIBVER(version)

In the following, each function is defined in detail. As seen, all DLL functions that involve operation of the DMX line are lpt-parametric, i.e. they accept the lpt port address as an input parameter. This allows simple support of up to three DMX interfaces on the three architected and supported LPT ports (base address 0x278, 0x378, 0x3BC).

2.1 - DMXINIT

The **DMXINIT** function is called once at program initialization in order to setup the DLL variables associated with the specified lpt interface.
The **lpt** parameter specifies the lpt port of the DMX interface.
The **dmxsize** parameter is used to setup the size of the buffer used by the DLL to handle DMX channel values, and is normally set at 512 for all selected lpt ports.
The **dmxmst** parameter is used to setup the driving state (master vs. slave) of the DMX interface. Master mode is selected with dmxmst=1 and Slave Mode with dmxmst=0.
Setting dmxmst=1 (Master Mode) specifies an internal active driving state that is used by the DLL to modify the execution of some functions, and it does not have any effect on the actual driving state of the DMX line, that is instead controlled by DMXDRIVE.
For example, when not in active driving state (i.e. when in Slave Mode), the DLL fakes (i.e. no-op) DMXSEND call, allowing the high-level program to be simpler.

Here is the function template:

```
;************************************************************************
;DMXINIT (lpt,dmxsize,dmxmst)
;
;function:      Initializes DMXLIB module variables: to be issued before
;                    issuing any other DMX command
;
;C++ prototype: extern __stdcall DMXINIT(int *, int *, int *);
;VB4 prototype: Declare Sub DMXINIT(lpt,dmxsize,dmxmst);
;
;input parameters:
;
;  lpt             32-bit pointer to Lpt base address
;  dmxsize      32-bit pointer to DMX string size (1 to 512 channels)
;  dmxmst       32-bit pointer to DMX Master/Slave mode (1=Master, 0=Slave)
;
;output parameters: none
;
;************************************************************************
```

2.2 - DMXDRIVE

The **DMXDRIVE** function is used to enable the DMX interface to drive the DMX line.
The **lpt** parameter is used to specify the port where the DMX interface is connected.
The **dmxstart** parameter is used to set the Startcode Byte as defined by DMX-512/1990 and is normally set at 0 for standard DMX devices like dimmers, scrollers and scanners.
The **dmxstatus** is a return value that is used to verify normal execution of the function. When DMXDRIVE execution is successful, the return value of dmxstatus must be 0. Any value different than 0 should be flagged to the user, indicating problems in the connection of the DMX interface that result in command timeout.
Prior to DMXDRIVE call, the DMXRELAX function may be called to select relaxed DMX timings (see next chapters).

```
;*****************************************************************************
;DMXDRIVE(lpt,dmxstart,dmxstatus)
;
;function:          Issue Drive command to ParDmx to prepare for DMXSend
;                   or DMXPtcSnd; set DrvAct variable on completion.
;                   Check for timeout during hardware handshake due to
;                   fatal hardware errors of ParDmx interface
;
;C++ prototype: extern __stdcall DMXDRIVE(int*, int *, int *);
;VB4 prototype: Declare Sub DMXDRIVE(lpt,dmxstart,dmxstatus);
;
;input parameters:
;
;          lpt              32-bit pointer to Lpt base address
;
;          dmxstart         32-bit pointer to dmxstart (DMX Start Code)
;
;output parameters:
;
;          dmxstatus        32-bit pointer to dmxstatus
;                                   dmxstatus: (<>0000h=timeout, 0000h=successful)
;
;*****************************************************************************
```

2.3 - DMXRELAX

The **DMXRELAX** function is used to command the DMX interface to use relaxed timings on the DMX line.
The **lpt** parameter is used to specify the port where the DMX interface is connected.
This function is typically useful when working with critical or marginal DMX-512 equipment that cannot tolerate the maximum theoretical speed and may therefore need relaxed timing. Here is how typical timings are altered (values for Digimedia DMXADP Compact Parallel Port Adapter):

Parameter	**unit**	**DMXADP Standard Timing**	**DMXADP Relaxed Timing**
		(standard)	(relaxed)
Break	usec	101-108	198-205
Mab	usec	17-21	26-30
Byte/pack		512	512
Break-to-Break	usec	24742	28135
Updates/sec		40-41	35-36

```
;*****************************************************************************
;_DMXRELAX(lpt)
;
;function:          Set Relaxed Timings on DMX line. Changes are effective on next ;
          DMXDRIVE function.
;
;C++ prototype: extern "C" DMXRELAX(int *);
;VB4 prototype: Declare Sub DMXRELAX (lpt);
;
;input parameters:
;
;          lpt                  32-bit pointer to LPT port address
;
;output parameters:
;
;          none
;
;*****************************************************************************
```

2.4 - DMXSTANDARD

The **DMXSTANDARD** function is used to command the DMX interface to use standard timings on the DMX line. This function may be used to eliminate the effect of a previously issued DMXRELAX function.
The **lpt** parameter is used to specify the port where the DMX interface is connected. Standard DMX timing values are restored on the next call to DMXDRIVE therefore a DMXSTANDARD/DMXDRIVE sequence is required to reset a DMXRELAX/DMXDRIVE sequence.

```
;*****************************************************************************
;_DMXSTANDARD(lpt)
;
;function:          Set Standard Timings on DMX line. Changes are effective on next ;
                    DMXDRIVE function.
;
;C++ prototype: extern "C" DMXSTANDARD(int *);
;VB4 prototype: Declare Sub DMXSTANDARD (lpt);
;
;input parameters:
;
;          lpt                32-bit pointer to LPT port address
;
;output parameters:
;
;          none
;
;*****************************************************************************
```

2.5 - DMXSENSE

The DMXSENSE function is used to check the presence of the DMX interface and it reports some information on the interface type.
The **lpt** parameter is used to specify the port where the DMX interface is connected.
The **dmxstatus** return parameter is used to report cumulative errors and should always be returned at 0. If different than zero, the value should be reported to the user for troubleshooting.
The **dmxtype** return parameter is used to repor a two-state condition. With DMXPlus, this is used to report which model is connected (the PRO or the BASE model, now discontinued by Digimedia). With DMXAdp, this is used to report which cable is connected (regular cable or Intelligent Cable) and let the program prompt the user for manual termination handling when the standard cable is used.
Note that to simplify the use of the high-level program with no DMX interface connected (e.g. for off-line show preparation), the DLL enters an automatic demo mode when the first DMXSENSE call executed after DMXINIT returns command timeout. In this case all the following function calls are skipped to let the program run unchanged with no hardware connected and no timeout overhead. The type of interface connected, including the no-hardware demo case is reported in detail by the DMXIDENT function.

```
;************************************************************************
;DMXSENSE(lpt,dmxstatus,dmxtype)
;
;function:      Issues Sense command to ParDmx to test if it is present
;                 and what type of hardware is present
;                 and resets DmxPres flag only when called first time
;                 (checked via FirstSns flag) if hardware not sensed
;               Function skipped with ok return code if DmxPres flag inactive
;                 to allow automatic demo mode of periodic Sense call
;               Reports cumulative status information about previous
;                 silent errors on DMXSend or DMXPtcSnd to support periodic
;                 off-line error test
;               First DMXSense call to be done after DMXInit before any
;                 other function call
;
;C++ prototype: extern __stdcall DMXSENSE(int*, int *, int *);
;VB4 prototype: Declare Sub DMXSENSE(lpt,dmxstatus,dmxtype);
;
;input parameters:
;
;         lpt              32-bit pointer to Lpt address
;
;output parameters:
;
;         dmxstatus        32-bit pointer to DMXStatus
;                          DMXStatus (0000h=ParDmx present, no errors logged
;                            xxyy=errors where:
;                               xx<>0 = Sense failed  yy<>0 = logged errors
;
;         dmxtype          32-bit pointer to DMXType (valid on DMXStatus=0)
;                          for DMXPlus: 0=Base 1=Pro
;                          for DMXAdp: 0=std cable 1=Intelligent Cable
;
;************************************************************************
```

2.6 - DMXCAPTURE

The DMXCAPTURE function is used to enter Slave Mode (if previously driving DMX line) and capture the first packet detected on the DMX line. In this way the state of the DMX line may be monitored and static scenes may be captured and stored in the high-level program. The capturing performance is in the range of 15 packets/sec, therefore it is good enough for a realistic DMX monitor but is not meant for capture/playback of crossfades from an external console.
As usually, the **lpt** parameter specifies the port where the DMX interface is plugged.
The **dmxstart** return parameter reports the value of the DMX Startcode of the captured packet (normally 0 fro dimmer applications). The **dmxpkt** value reports the acquired number of channels from 0 to 512, where a value of 0 indicates a capture timeout (i.e. no DMX signal found).
The **dmxchan** parameter is a pointer to the channel buffer, that must be an array of 32-bit words.

```
;************************************************************************
;DMXCAPTURE(lpt,dmxstart,dmxpkt,dmxchan)
;
;function:      Captures a packet from ParDmx interface after issuing a
;                   reverse line command. Captures all DMX-512 startcodes
;                   and return startcode. Packet size is adaptive and
;                   returned to calling program. Checks handshake timeouts
;                   and returns DMXPkt=0 on any error. If a hardware command
;                   handshake error is detected, error is logged in DmxErr and
;                   will be reported as fatal by the next DMXSense call
;
;C++ prototype: extern __stdcall DMXCAPTURE(int *, int *, int *, int *);
;VB4 prototype: Declare Sub DMXCAPTURE(lpt,dmxstart,dmxpkt,dmxchan);
;
;
;input parameters:
;
;  lpt                  32-bit pointer to Lpt base address
;  dmxchan       32-bit pointer to DMX channels array (max 512 words)
;
;output parameters:
;
;  dmxstart     32-bit pointer to startcode of received DMX packet
;  dmxpkt       32-bit pointer to received DMX packet lenght
;
;************************************************************************
```

2.7- DMXSEND

The DMXSEND function is simply user to update the channel values to be transmitted on the DMX line. All DMX interfaces designed by Digimedia are intelligent, and perform automatic refresh of the DMX line with the last transferred values. All crossfades are computed by the PC CPU and transferred to the interface with the DMXSEND function.
The **lpt** parameter specifies the parallel port.
The **dmxchan** parameter is a pointer to the channel buffer, that must be an array of 32-bit words.

```
;************************************************************************
;DMXSEND(lpt,dmxchan)
;
;function:      send DMX channel values for pre-configured size onto
;               ParDmx interface
;                       in case of handshake timeout the function is skipped
;                       and an error is logged on DmxErr variable
;
;C++ prototype: extern __stdcall DMXSEND(int *);
;VB4 prototype: Declare Sub DMXSEND(lpt,dmxchan());
;
;input parameters:
;
;  lpt          32-bit pointer to Lpt base address
;  dmxchan      32-bit pointer to DMX channels array (max 512 dwords)
;
;output parameters: none
;
;************************************************************************
```

2.8 - DMXPTCSND

The DMXPTCSND ("Patch-Send") is a structured version of the DMXSEND designed to offload the high-level program from some channel-management tasks that are typical of lighting consoles. Being entirely written in machine-language, the DLL is much faster at perfroming these functions than the high-level calling program.
The **lpt** and **dmxchan** parameters have the same meaning of the DMXSEND function.
The **dmxchan1**, **dmxchan2**, **dmxchan3**, **dmxchan4** are pointers to four additional channel buffers that operates in the HTP (Highest Takes Priority) fashion to implement high-level functions like chasers. The **dmxgr** is a pointer to a "Group" buffer, that is used to implement high-priority registers.
The **mast** parameter is the "Grand Master" and it allows local master control (0-100%) performed automatically by the DLL. This is done on all channels that do not have "Mater-Inhibit" curves as explained in the following.
The **dmxpatch**, **dmxpatchlev** are pointers to two arrays that perform the modulated-patch function. In this way a channel may be attributed to a dimmer with a specific level (0-200%) that allows adjusting dimmer levels to lamp chracteristics maintaining the show unchanged.
Finally, the **dmxcurve** specifies the type of curve used to control the dimmer, with this convention:

0	linear
1	ON-OFF (dimmer off if zero, 100 if different than 0)
2	first channel of a 16-bit circuit (for high-level program use)
3	constantly at patch level (independently from master and black-outs)

Adding 10 the Mater-inhibitor is selected as follows:

10	linear + No-Master
11	ON-OFF + No-Master
12	first channel of a 16-bit circuit + No-Master

This is a pseudo-code representation of the DMXPTCSND function flow:

```
FOR i = 1 TO dmxsize
dmxch(i) = MAX(dmxchan(dmxpatch(i)), dmxchan1(dmxpatch(i)),
               dmxchan2(dmxpatch(i)), dmxchan3(dmxpatch(i)),
               dmxchan4(dmxpatch(i)), dmxgr(dmxpatch(i)))
IF dmxcurve(i) < 10 THEN dmxch(i) = dmxch(i) * (mast / 100)
IF dmxcurve(i) = 3 THEN dmxch(i) = 255
dmxch(i) = (dmxch(i) * (dmxpatchlev / 100))
IF dmxch(i) > 255 THEN dmxch(i) = 255
IF dmxcurve(i) = 1 AND dmxch(i) <> 0 THEN dmxch(i) = 255
NEXT i
CALL DMXSEND(lpt,dmxch(1))
```

The algorithm shown allows sophisticated channel control to be automatically performed by the DLL with no burden on the high-level program other than the compilation of the channels, patch and curve arrays.

The PTCSND function prototype is reported in the following:

```
;************************************************************************
;DMXPTCSND(lpt,mast,dmxchan,dmxchan1,dmxchan2,dmxchan3,dmxchan4,
;                   dmxgr,dmxpatch,dmxpatchlev,dmxcurve)
;
;function:          send DMX channel values for pre-configured size onto
;                   ParDmx interface, applying Master, Group, Patch,
;                   PatchLevel and Curve parameter
;                   in case of hahdshake timeout the function is skipped
;                   and an error is logged on DmxErr variable
;
;input parameters:
;
;          lpt               32-bit pointer to LPT port address (integer)
;          mast              32-bit pointer to master value (integer)
;          dmxchan           32-bit pointer to DMX channels array   (512 dwords)
;          dmxchan1          32-bit pointer to DMX channels array 1 (512 dwords)
;          dmxchan2          32-bit pointer to DMX channels array 2 (512 dwords)
;          dmxchan3          32-bit pointer to DMX channels array 3 (512 dwords)
;          dmxchan4          32-bit pointer to DMX channels array 4 (512 dwords)
;          dmxgr             32-bit pointer to DMX Group array (512 dwords)
;          dmxpatch          32-bit pointer to DMX Patch array (512 dwords)
;          dmxpatchlev       32-bit pointer to DMX Patch Level array (512 dwords)
;          dmxcurve          32-bit pointer to DMX Curve array (512 dwords)
;
;output parameters: none
;
;************************************************************************
```

2.9 - DMXIDENT

The DMXIDENT function is used to report a cumulative set of information regarding the DMX interface hardware. This function is used to handle the historical differences between the different models of DMX hardware produced by Digimedia and supports different protection schemes to link software features to a specific model or serial number.
The usual **lpt** parameter specifies the parallel port.
The **dmxstatus** is the return variable indicating inquiry success (0 returned) or te occurrence of any problem during hardware inquiry.
The function, when successfully completed (dmxstatus=0) reports the information about the interface in the **dmxhard**, **dmxrev**, **dmxsubrev**, **serialno** variables in this way:

VARIABLE	MEANING	RETURNED VALUES
dmxhard	hardware type	0=none 1=DMXPlus Base 2=DMXPlus Pro 3=DMXAdp
dmxrev	firmware revision	1 Ascii character
dmxsubrev	firmware sub-revision	2 Ascii characters
serialno	serial number	5-digit BCD (nulls if no s/n)

This is the function prototype:

```
;************************************************************************
;DMXIDENT(lpt,dmxstatus,dmxhard,dmxrev,dmxsubrev,serialno)
;
;function:     Used to identify the DMX interface hardware connected to LPT
;                  setting dmxplus flag accordingly to support run-time selection
;                  of DMX hardware. This is the proper function sequence:
;
;                  DMXInit        to setup library
;                  DMXSense       to verify hardware presence
;                  DMXIdent       only if Sense successful
;
;input parameters:
;         lpt              32-bit pointer to Lpt Port address
;
;output parameters: DMXStatus (0000h=ParDmx present, no errors logged
;                                  xxyy=errors where:
;                                      xx<>0 = Sense failed
;                                      yy<>0 = logged Send-PtcSnd errors)
;
;         DMXHard          32-bit pointer to DMX Interface Hardware type encoding
;                                  0000h=no hardware (demo mode)
;                                  0001h=DMXPlus Base
;                                  0002h=DMXPlus Pro
;                                  0003h=DMXAdp
;
;         DMXRev           32-bit to Hardware Revision code (1 Ascii)
;;        DMXSubRev        32-bit pointer to Hardware SubRevision code (2 Ascii)
;
;         serialno         32-bit serial number in BCD format if present
;
;************************************************************************
```

2.10 - DMXREV

The DMXREV function is used to report the full revision template of the DMX interface firmware revision. This is normally not required since DMXIDENT is used to report firmware revision/subrevision. As with DMXIDENT, this function let the DMX interface be used as a hardware key. DMXREV may be only called when in Slave Mode, i.e. it must be called before the DMXDRIVE function. Calling DMXREV from Master mode clears the active driving state and require a DMXDRIVE command to be re-issued before making DMXSEND or DMXPTCSND calls.
The **lpt** parameter specifies the parallel port.
The **revsiz** return parameter specifies the number of revision bytes returned by the interface (equal to 30 for all present Digimedia hardware).
The **revbuf** parameter is a pointer to a revision buffer that must be at least 32 words-long.

The typical returned revision strings are in the following format:

DMXPlus:

```
1996:     no revision string
1997:     (C)DMXEASY'97-PLUS-sssss-Vr.rr
```

DMXAdp:

```
1996:     (C)DMXEASY 1996 - DMXADP Vr.rr
1997:     (C)DMXEASY'97-ADP -sssss-Vr.rr
1998:     ®DIGIMEDIA'98-ADP -sssss-Vr.rr
```

Where "r.rr" is a revision/subrevision field (e.g. 1.71 or 3.10) and "sssss" is a 5-digit serial number (e.g. 00102).

This is the function prototype:

```
;************************************************************************
;DMXREV(lpt,revsiz,revbuf)
;
;function:          Captures a revision-level packet from DMXADP
;                   Checks handshake timeouts and returns DMXPkt=0 on any error
;
;
;C++ prototype: extern __stdcall DMXREV(int *, int *);
;
;
;input parameters: none
;
;  lpt         32-bit pointer to Lpt base address
;
;output parameters:
;
;  revsiz     32-bit pointer to size of received rev packet
;  revbuf     32-bit pointer to received rev packet (size <= 16bytes)
;
;************************************************************************
```

2.11 - LIBVER

The LIBVER function is simply used to report the revision level of the DMXPLUS.DLL library to the high-level calling program. This is used to report a global configuration to the user and simplify field maintenance.
The only returned parameter is **version**, that is a ASCII-packed word in the x.yz format (e.g. 6.50).

```
;************************************************************************
;LIBVER(version)
;
;function:      return DMX library version
;
;C++ prototype: extern __stdcall LIBVER(int *);
;VB4 prototype: Declare Sub LIBVER(version);
;
;input parameters:
;
;  none
;
;output parameters: none
;
;  version         32-bit pointer to version structure x.yz ASCII
;
;************************************************************************
```

3 - Using DMXPLUS.DLL functions

Three applications of DMXPLUS.DLL are described herein. Granted OEM users may get additional information on the DLL contacting technical support at Digimedia using the email (**support @digimedia-mls.com**) or website (**http://www.digimedia-mls.com**) entry points.

3.1 - Simple Master-only program

The simplest application is a program that only sends DMX to control lighting equipment. Following Win32 specified STDCALL calling convention, all parameters are far (e.g 32-bit) pointers.

The first information to determine is the base address of the LPT port used for the DMX interface. A set of different techniques may be used, from user option box to automatic identification. The simplest one is a static LPT port setup, that may be modified with a configuration menu.
As a reminder, note that the LPT1, LPT2, LPT3 port identifier used by Microsoft are not in direct relationship with the base address. The three architected base adresses for the IBM-compatible PCs are 0x278, 0x378 (most likely the default for single-port PCs) and 0x3BC.
Most BIOS setup procedures include LPT base address and display the base address when booting the machine. Under Win95, the base address may be obtained using the Configuration Panel/System/Port/Properties menu.
Note that DMXPLUS.DLL uses the simplest mode of the LPT port, also called **SPP** for Standard Parallel Port, and it does not use the PS/2 bidirectional or the EPP enhanced mode. Therefore any LPT port may be used for current Digimedia products, guaranteeing total compatibility with any parallel port interface.

Here is the simplest sequence for a DMX send application in pseudo-C++ format:

```
lpt = 0x378; // set depending on selected LPT port //
dmxsize = 512;
dmxmst = 1;    // set internal driving active mode  //
dmxstart = 0;  // startcode for DMX-512/1990 dimmers //

DMXINIT (&lpt,&dmxsize,&dmxmst); // setup DLL variables //

DMXDRIVE(&lpt,&dmxstart,&dmxstatus); // enable DMX line driver //
if (dmxstatus != 0)
   printf("\n    Drive command error, status %x \n", dmxstatus);
else
   printf("\nDrive command successful");

// set DMX channel values in dmxchan buffer //

DMXSEND(&lpt,&dmxchan[0]); // send new channels on DMX line   //
               // last packet sent is automatically refreshed //
               // on DMX line by DMXADP hardware              //
```

3.2 - Master-only program with identification and patch-send

A more complete Master program that identifies hardware and uses the complete Patch-Send functions would use the following flow:

```
lpt = 0x378; // set depending on selected LPT port //
dmxsize = 512;
dmxmst = 1;    // set internal driving active mode  //
dmxstart = 0;  // startcode for DMX-512/1990 dimmers //
mast = 100;    // set Grand Master at full //

DMXINIT (&lpt,&dmxsize,&dmxmst); // setup DLL variables //
DMXSENSE (&lpt,&dmxstatus,&dmxtype); // first sense //
DMXIDENT (&lpt,&dmxstatus,&dmxhard,&dmxrev,&dmxsubrev,&serialno);

// report dmxhard, dmxrev, dmxsubrev, serialno to user //

DMXDRIVE(&lpt,&dmxstart,&dmxstatus); // enable DMX line driver //
if (dmxstatus != 0)
   printf("\n    Drive command error, status %x \n", dmxstatus);
else
   printf("\nDrive command successful");

// set default array values //

for (i=0; i < dmxsize; i++)
   {
   dmxchan[i] = 0;       // setup channel at zero //
   dmxchan1[i] = 0;      // setup chaser1 at zero //
   dmxchan2[i] = 0;      // setup chaser2 at zero //
   dmxchan3[i] = 0;      // setup chaser3 at zero //
   dmxchan4[i] = 0;      // setup chaser4 at zero //
   dmxpatch[i] = i;      // default patch 1:1 //
   dmxpatchlev[i] = 100; // default patchlevel full //
   dmxcurve[i] = 0       // default curve linear+master //
   }

for (i=0; i < dmxsize+1; i++)
   {
   dmxgr[i] = 0;         // setup groups at zero //
   }

// set DMX channel values in dmxchan buffer //
// set master and individual circuit patch, curve and level //

DMXPTCSND(&mast, &dmxchan[0], &dmxchan1[0], &dmxchan2[0],
            &dmxchan3[0], &dmxchan4[0],&dmxgr[0], &dmxpatch[0],
            &dmxpatchlev[0], &dmxcurve[0]);  // patch-send //
```

3.2 - Master-Slave program

A simple application that switches between Master and Slave modes is reported in the following.
Note that, different than with DMXPlus hardware, the DMXAdp low-cost compact interface allows dynamic automatic switch between master and slave mode only with the Intelligent Cable option. In fact, to avoid conflicts when using the standard cable with DMXAdp, the DMX-IN signal should be disconnected and a termination inserted when switching from Slave to Master Mode.
Using the values returned by DMXSENSE and DMXIDENT, the program may detect the configuration (DMXAdp and standard cable) and warn the user to cable the line accordingly. While this is suitable for unfrequent Master-Mode use by mostly-Slave applications (e.g a DMX monitor), all applications requiring hot-key, on-the-fly switch between Mater and Slave will practically require the intelligent cable option of the most powerful DMXPlus hardware. This case is reported in the following example:

```
lpt = 0x378; // set depending on selected LPT port //
dmxsize = 512;
dmxmst = 0;    // clear internal driving active mode  //
dmxstart = 0;  // startcode for DMX-512/1990 dimmers //

DMXINIT (&lpt,&dmxsize,&dmxmst); // setup DLL variables //
DMXSENSE (&lpt,&dmxstatus,&dmxtype); // first sense //
DMXIDENT (&lpt,&dmxstatus,&dmxhard,&dmxrev,&dmxsubrev,&serialno);

// report dmxhard, dmxrev, dmxsubrev, serialno to user //

// perform the following capture in loop as required //

DMXCAPTURE(&lpt, &dmxstart, &dmxpkt, &dmxchan[0]);

if (dmxpkt != 0)
   {
   // handle captured channel values //
   }
else
   {
   // handle timeout (no DMX signal) case //
   }

// when desired, switch to Master Mode as follows //

DMXDRIVE(&lpt,&dmxstart,&dmxstatus); // enable DMX line driver //
if (dmxstatus != 0)
   printf("\n    Drive command error, status %x \n", dmxstatus);

// set DMX channel values in dmxchan buffer //

DMXSEND(&lpt,&dmxchan[0]); // send channels values to DMX line //

// back to DMXCAPTURE loop at any time to return to Slave Mode //
```

APPENDIX A – Using DMXPLUS.DLL in different languages

A.1 – Delphi

```
uses
  Messages, SysUtils, Classes, Controls,Dialogs,
  StdCtrls, Buttons, ExtCtrls;


   procedure EstablishDMXLink;
   procedure INITDMX;
   procedure SendDMX;

implementation

uses Unit1;


var  DMXSIZE,DMXMST,DMXSTART, DMXTYPE,
     DMXHARD, DMXREV, DMXSUBREV, SERIALNO : PChar;
     LPTENABLED                           : array [0..2] of Boolean;
     LPT,DMXCHAN,DMXSTATUS                : array [0..2] of PChar;


procedure DMXINIT(lpt, dmxsize, dmxmst: PChar); stdcall; external 'dmxplus.dll';
procedure DMXDRIVE(lpt, dmxstart, dmxstatus: PChar); stdcall; external
'dmxplus.dll';
procedure DMXSENSE(lpt, dmxstatus, dmxtype: PChar); stdcall; external
'dmxplus.dll';
procedure DMXSEND(lpt, dmxchan: PChar); stdcall; external 'dmxplus.dll';
procedure DMXIDENT(lpt, dmxstatus, dmxhard, dmxrev, dmxsubrev, serialno: PChar);
stdcall; external 'dmxplus.dll';

procedure EstablishDMXLink;

var Dmxtmp : Word;

begin
     LPTENABLED[0] := false;
     LPTENABLED[1] := false;
     LPTENABLED[2] := false;

     GetMem(DMXSIZE,4);
     GetMem(DMXMST,4);
     GetMem(LPT[0],4);
     GetMem(LPT[1],4);
     GetMem(LPT[2],4);
     GetMem(DMXSTART,4);
     GetMem(DMXSTATUS[0],4);
     GetMem(DMXSTATUS[1],4);
     GetMem(DMXSTATUS[2],4);
     GetMem(DMXTYPE,4);
     GetMem(DMXHARD,4);
     GetMem(DMXREV,4);
     GetMem(DMXSUBREV,4);
     GetMem(SERIALNO,4);

     GetMem(DMXCHAN[0],2048);
     GetMem(DMXCHAN[1],2048);
     GetMem(DMXCHAN[2],2048);
```

```
     For Dmxtmp := 0 to 3 do
     begin
          (DMXSIZE+Dmxtmp)^ := #0;
          (DMXMST+Dmxtmp)^ := #0;
          (LPT[0]+Dmxtmp)^ := #0;
          (LPT[1]+Dmxtmp)^ := #0;
          (LPT[2]+Dmxtmp)^ := #0;
          (DMXSTART+Dmxtmp)^ := #0;
          (DMXSTATUS[0]+Dmxtmp)^ := #0;
          (DMXSTATUS[1]+Dmxtmp)^ := #0;
          (DMXSTATUS[2]+Dmxtmp)^ := #0;
          (DMXTYPE+Dmxtmp)^ := #0;
     end;
     (DMXSIZE+1)^ := #2;
     (DMXMST)^ := #1;
     (LPT[0])^ := #120;
     (LPT[0] + 1)^ := #3;
     (LPT[1])^ := #120;
     (LPT[1] + 1)^ := #2;
     (LPT[2])^ := #188;
     (LPT[2] + 1)^ := #3;
     (DMXSTART)^ := #0;

end;

procedure InitDMX;
var Dmxtmp  : Word;

begin

     For Dmxtmp := 0 to 2047 do
     begin
          (DMXCHAN[0]+Dmxtmp)^ := #0;
          (DMXCHAN[1]+Dmxtmp)^ := #0;
          (DMXCHAN[2]+Dmxtmp)^ := #0;
     end;

     For Dmxtmp := 0 to 511 do
     begin
          If Unit1.Setup.Dmx[Dmxtmp].Ch div 512 = 0 then
          begin
               LPTENABLED[0]:=true;
          end;

          If Unit1.Setup.Dmx[Dmxtmp].Ch div 512 = 1 then
          begin
               LPTENABLED[1]:=true;
          end;

          If Unit1.Setup.Dmx[Dmxtmp].Ch div 512 = 2 then
          begin
               LPTENABLED[2]:=true;
          end;
     end;


     If LPTENABLED[0] then DMXINIT(LPT[0],DMXSIZE,DMXMST);
     If LPTENABLED[1] then DMXINIT(LPT[1],DMXSIZE,DMXMST);
     If LPTENABLED[2] then DMXINIT(LPT[2],DMXSIZE,DMXMST);

     If LPTENABLED[0] then
     begin
          DMXSENSE(LPT[0],DMXStatus[0],DMXType);
          If (DMXStatus[0]^ = #0) and
             ((DMXStatus[0]+1)^ = #0) and
             ((DMXStatus[0]+2)^ = #0) and
             ((DMXStatus[0]+3)^ = #0)
```

```
          then LPTENABLED[0] := true
          else LPTENABLED[0] := false;
          If LPTENABLED[0] then DMXIDENT(LPT[0], DMXSTATUS[0], DMXHARD, DMXREV,
DMXSUBREV, SERIALNO);
     end;
     If LPTENABLED[1] then
     begin
          DMXSENSE(LPT[1],DMXStatus[1],DMXType);
          If (DMXStatus[1]^ = #0) and
             ((DMXStatus[1]+1)^ = #0) and
             ((DMXStatus[1]+2)^ = #0) and
             ((DMXStatus[1]+3)^ = #0)
          then LPTENABLED[1] := true
          else LPTENABLED[1] := false;
          If LPTENABLED[1] then DMXIDENT(LPT[1], DMXSTATUS[1], DMXHARD, DMXREV,
DMXSUBREV, SERIALNO);
     end;
     If LPTENABLED[2] then
     begin
          DMXSENSE(LPT[2],DMXStatus[2],DMXType);
          If (DMXStatus[2]^ = #0) and
             ((DMXStatus[2]+1)^ = #0) and
             ((DMXStatus[2]+2)^ = #0) and
             ((DMXStatus[2]+3)^ = #0)
          then LPTENABLED[2] := true
          else LPTENABLED[2] := false;
          If LPTENABLED[2] then DMXIDENT(LPT[2], DMXSTATUS[2], DMXHARD, DMXREV,
DMXSUBREV, SERIALNO);
     end;

     If LPTENABLED[0] then DMXDRIVE(LPT[0],DMXSTART,DMXSTATUS[0]);
     If DMXSTATUS[0]^ <> #0 then
     begin
          LPTENABLED[0] := false;
     end;

     If LPTENABLED[1] then DMXDRIVE(LPT[1],DMXSTART,DMXSTATUS[1]);
     If DMXSTATUS[1]^ <> #0 then
     begin
          LPTENABLED[1] := false;
     end;

     If LPTENABLED[2] then DMXDRIVE(LPT[2],DMXSTART,DMXSTATUS[2]);
     If DMXSTATUS[2]^ <> #0 then
     begin
          LPTENABLED[2] := false;
     end;

     For DmxTmp := 0 to 511 do
     begin
          If Unit1.Setup.DMX[DmxTmp].Ch div 512 = 0 then
             (DMXCHAN[0]+4*Unit1.Setup.DMX[DmxTmp].Ch)^ :=
chr(Unit1.Setup.Dmx[DmxTmp].Preheat);
          If Unit1.Setup.DMX[DmxTmp].Ch div 512 = 1 then
             (DMXCHAN[1]+4*Unit1.Setup.DMX[DmxTmp].Ch)^ :=
chr(Unit1.Setup.Dmx[DmxTmp].Preheat);
          If Unit1.Setup.DMX[DmxTmp].Ch div 512 = 2 then
             (DMXCHAN[2]+4*Unit1.Setup.DMX[DmxTmp].Ch)^ :=
chr(Unit1.Setup.Dmx[DmxTmp].Preheat);
     end;

     If LPTENABLED[0] then DMXSEND(LPT[0],DMXCHAN[0]);
     If LPTENABLED[1] then DMXSEND(LPT[1],DMXCHAN[1]);
     If LPTENABLED[2] then DMXSEND(LPT[2],DMXCHAN[2]);

end;
```

```
procedure SendDMX;
var Dmxtmp : Word;

begin
     For DmxTmp := 0 to 511 do
     begin
          If Unit1.Setup.DMX[DmxTmp].Ch div 512 = 0 then
             (DMXCHAN[0]+4*Unit1.Setup.DMX[DmxTmp].Ch)^ :=
(Unit1.P+512*Pointer+DmxTmp)^;
          If Unit1.Setup.DMX[DmxTmp].Ch div 512 = 1 then
             (DMXCHAN[1]+4*Unit1.Setup.DMX[DmxTmp].Ch)^ :=
(Unit1.P+512*Pointer+DmxTmp)^;
          If Unit1.Setup.DMX[DmxTmp].Ch div 512 = 2 then
             (DMXCHAN[2]+4*Unit1.Setup.DMX[DmxTmp].Ch)^ :=
(Unit1.P+512*Pointer+DmxTmp)^;
     end;

     If Unit1.Max > 0 then
     begin
          If Unit1.Setup.DMX[Unit1.Max-1].Ch div 512 = 0 then
             (DMXCHAN[0]+4*(Unit1.Setup.DMX[Unit1.Max-1].Ch))^ := Chr(255);
          If Unit1.Setup.DMX[Unit1.Max-1].Ch div 512 = 1 then
             (DMXCHAN[1]+4*(Unit1.Setup.DMX[Unit1.Max-1].Ch))^ := Chr(255);
          If Unit1.Setup.DMX[Unit1.Max-1].Ch div 512 = 2 then
             (DMXCHAN[2]+4*(Unit1.Setup.DMX[Unit1.Max-1].Ch))^ := Chr(255);
     end;

     If LPTENABLED[0] then DMXSEND(LPT[0],DMXCHAN[0]);
     If LPTENABLED[1] then DMXSEND(LPT[1],DMXCHAN[1]);
     If LPTENABLED[2] then DMXSEND(LPT[2],DMXCHAN[2]);

end;

end.
```

A.2 – Visual C++

```
BOOL dmxeasy_ok = FALSE;
HINSTANCE hLibrary;
FARPROC lpFonction;

int lpt,
      dmxsize,
      dmxmst,
      dmxstart,
      dmxstatus;

BOOL Init_dmxeasy()
{
      lpt = 0x378; // set depending on selected LPT port //
      dmxsize = 512;
      dmxmst  = 1;    // 1=SEND 0=CAPTURE set internal driving active mode
//
      dmxstart = 0;     // startcode for DMX-512/1990 dimmers //

      hLibrary = LoadLibrary("dmxplus");
    if (hLibrary == NULL)
          return FALSE;

      lpFonction = GetProcAddress(hLibrary, "DMXINIT");
      (*lpFonction) (&lpt,&dmxsize,&dmxmst); // setup DLL variables //
      //DMXINIT (&lpt,&dmxsize,&dmxmst); // setup DLL variables //

      lpFonction = GetProcAddress(hLibrary, "DMXDRIVE");
      (*lpFonction) (&lpt,&dmxstart,&dmxstatus); // setup DLL variables //
      //DMXDRIVE(&lpt,&dmxstart,&dmxstatus); // enable DMX line driver //

      if (dmxstatus != 0)
         return FALSE;

      return TRUE;
}

void dmxeasy(unsigned char *bloc, int canaux)
{
      int dmxchan[512],i;

      for(i=0;i<512;i++)
        if (i<canaux)
            dmxchan[i] = (int) bloc[i];
        else
            dmxchan[i] = (int) 0;

      lpFonction = GetProcAddress(hLibrary, "DMXSEND");
      (*lpFonction) (&lpt,&dmxchan[0]); // setup DLL variables //
      //DMXSEND(&lpt,&dmxchan[0]); // send new channels on DMX line   //
                        // last packet sent is automatically refreshed //
                        // on DMX line by DMXADP hardware              //
}
```

A.3 – Visual Basic

```
'******************************
'
'Definitions in main BAS module
'
'******************************

'*********************
'variables declaration
'*********************

Public lpt As Long
Public dmxsize  As Long
Public dmxmst As Long
Public dmxstatus As Long
Public dmxhard As Long
Public dmxrv As Long
Public dmxsbrv As Long
Public dmxstart As Long
Public dmxpkt As Long
Public dmxtype As Long
Public revsiz As Long
Public mast As Long
Public version As Long
Public serialno As Long


'******************
'arrays declaration
'******************

Public revbuf(32) As Long
Public dmxchan(512) As Long
Public revstr(32) As String
Public verstr(4) As String

'**************************
'DLL subroutines declaration
'**************************

Declare Sub DMXINIT Lib "dmxplus.dll" (ByRef lpt As Long, ByRef dmxsize As Long,
ByRef dmxmast As Long)
Declare Sub DMXSENSE Lib "dmxplus.dll" (ByRef lpt As Long, ByRef dmxstatus As
Long, ByRef dmxtype As Long)
Declare Sub DMXRELAX Lib "dmxplus.dll" (ByRef lpt As Long)
Declare Sub DMXSTANDARD "dmxplus.dll" (ByRef lpt As Long)
Declare Sub DMXDRIVE Lib "dmxplus.dll" (ByRef lpt As Long, ByRef dmxstart As
Long, ByRef dmxstatus As Long)
Declare Sub DMXSEND Lib "dmxplus.dll" (ByRef lpt As Long, ByRef dmxchan As Long)
Declare Sub DMXREV Lib "dmxplus.dll" (ByRef lpt As Long, ByRef revsiz As Long,
ByRef revbuf As Long)
Declare Sub DMXPTCSND Lib "dmxplus.dll" (ByRef lpt As Long, ByRef mast As Long,
ByRef dmxchan As Long, ByRef dmxchan1 As Long, ByRef dmxchan2 As Long, ByRef
dmxchan3 As Long, ByRef dmxchan4 As Long, ByRef dmxgr As Long, ByRef dmxpatch As
Long, ByRef dmxpatchlev As Long, ByRef dmxcurva As Long)
Declare Sub DMXIDENT Lib "dmxplus.dll" (ByRef lpt As Long, ByRef dmxstatus As
Long, ByRef dmxhard As Long, ByRef dmxrv As Long, ByRef dmxsbrv As Long, ByRef
serialno As Long)
Declare Sub DMXCAPTURE Lib "dmxplus.dll" (ByRef lpt As Long, ByRef dmxstart As
Long, ByRef dmxpkt As Long, ByRef dmxchan As Long)
Declare Sub LIBVER Lib "dmxplus.dll" (ByRef version As Long)


.........
```

```
'*****************************
'
'PROGRAM in form or module code
'
'*****************************

'***************
'variables setup
'***************

dmxsize = 512
dmxmst = 1
dmxstart = 0
dmxstatus = 0
dmxhard = 0
dmxrv = 0
dmxsbrv = 0
dmxsize = 512
dmxpkt = 0
dmxtype = 0
revsiz = 0
mast = 100
version = 0
serialno = 0

'****************************
'DMX Interface initialization
'****************************

'default LPT assignment/init
lpt = &H378
'initialize DMX interface
Call DMXINIT(lpt, dmxsize, dmxmst)


'************************************
'simple DMX send, all channels to zero
'************************************


'case Standard DMX timings (default)
'Issue Drive command to enable DMX output
Call DMXDRIVE(lpt, dmxstart, dmxstatus)
Debug.Print "status=" + Str(dmxstatus)

'case Relaxed DMX timings (alternative)
Call DMXRELAX(lpt)
'Issue Drive command to enable DMX output
Call DMXDRIVE(lpt, dmxstart, dmxstatus)
Debug.Print "status=" + Str(dmxstatus)


'setup channels
For J% = 1 To dmxsize
dmxchan(J%) = 0
Next J%

'send DMX channels values to DMX
Call DMXSEND(lpt, dmxchan(1))

……
```

```
'******************
'simple DMX receive
'******************

'setup array to pre-receive data to check update from DMX
For J% = 1 To dmxsize
dmxchan(J%) = &H55
Next J%

Call DMXCAPTURE(lpt, dmxstart, dmxpkt, dmxchan(1))

Debug.Print "Received " + Str(dmxpkt) + " channels from StartCode " +
Str(dmxstart)
Debug.Print "Capture Complete"
'captured channels are in dmxchan(i) array

……

'*************************
'Get DMX interface revision
'*************************

Call DMXREV(lpt, revsiz, revbuf(1))

For i% = 1 To 32
revstr(i%) = ""
Next i%
For i% = 1 To revsiz
revstr(i%) = Chr$(revbuf(i%) Mod 256)
Next i%

……

'**********************
'Identify DMX Inteface
'**********************

Call DMXIDENT(lpt, dmxstatus, dmxhard, dmxrv, dmxsbrv, serialno)

If (dmxrv <> 0) And (dmxsbrv <> 0) Then
Debug.Print "HW (0:none/1:DMXPlus Base/2:DMXPlus Pro/3:DMXADP)=" + Str(dmxhard)
+ " RV=" + Chr$(dmxrv) + "." + Chr$(dmxsbrv \ 256) + Chr$(dmxsbrv Mod 256) + "
sn=" + Hex$(serialno)
End If

……


'***************
'get DLL Version
'***************

Call LIBVER(version)

'extract version string from zy.x byte string
For i% = 1 To 4
verstr(i%) = Chr$(version Mod 256)
version = version \ 256
Next i%

Debug.Print "Library Version = " + verstr(1) + verstr(2) + verstr(3) + verstr(4)
```